UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BARCELONA TECH

Disseny i programació d'un sistema intel.ligent per control d'accessos

Autor: Jordi Cebrian Alonso
Director: Enric Xavier Martin Rull
Departament: Enginyeria de Sistemes, Automàtica i Informàtica Industrial (ESAII)

Grau en Enginyeria Informàtica
Especialitat en Enginyeria de Computadors
FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

Índex de continguts

1. Introducció

Avui en dia existeixen tot tipus de sistemes de reconeixement d'accés a un habitatge amb multitud de mides i funcionalitats. Aquest projecte pretén millorar els sistemes més comuns d'accés a un habitatge els quals incorporen una matriu de botons acompanyats d'una càmera, un micròfon i un altaveu.

L'objectiu és obtenir un sistema portable i compacte el qual eliminarà la matriu de botons dels sistemes convencionals i incorporarà un mòdul de reconeixement de veu per tal de dir a viva veu a quin pis es vol accedir i la transmissió de les dades anirà única i exclusivament mitjançant una xarxa sense fils.

Per tal d'assegurar la transmissió de les dades en temps real s'utilitzarà el protocol RTSP (Real Time Streaming Protocol).

1.1. Abast

El projecte consisteix en el disseny i la programació d'un sistema intel·ligent que controli els accessos a un habitatge.

El sistema constarà de dues parts:

- Mòdul extern: s'encarregarà mitjançant reconeixement de veu de detectar a quin pis es vol accedir i enviarà la veu i la imatge de la persona que vol accedir en temps real al pis en qüestió.

- Mòdul intern: serà el que estarà ubicat en un pis en qüestió i rebrà l'avís que s'està intentant accedir a l'habitatge i a aquell pis en concret i també rebrà la imatge i la veu de la persona que vol accedir. La comunicació per veu serà bidireccional, és a dir, que la persona que viu en l'habitatge podrà parlar en temps real amb la persona que sol·licita l'accés però aquesta última no podrà veure la imatge de la persona que hi ha en el pis. Aquesta última podrà rebutjar o permetre l'accés a l'habitatge. Aquesta segona part estarà emulada mitjançant un ordinador.

Les dues parts estaran interconnectades mitjançant una xarxa sense fils.

1.2. Estat de l’art

1.2.1. Plaques microprocessadores

Hi ha multitud de plaques microprocessadores al mercat, les més importants (pel seu reduït cost i la gran multitud de perifèrics que es poden connectar a aquestes) són les següents:

RaspBerry Pi

La **Raspberry Pi** és una placa microprocessadora de baix cost que inclou un SoC Broadcom BCM2835, que conté un processador central (CPU) ARM11 a 700 MHz. El firmware també inclou un mode Turbo per a que l'usuari pugui fer overclock fins a 1 GHz sense perdre la garantia, un processador gràfic (GPU) VideoCore IV, i 512 MB de memòria RAM (model B) encara que originalment en ser llançat eren 256 MB (model A). El disseny no inclou un disc dur o una unitat d'estat sòlid, ja que utilitza una targeta SD per a l'emmagatzematge permanent; tampoc inclou font d'alimentació o carcassa. El model B es ven a 35 €, i el model A a 25 €.

A la placa ens trobem a més amb una sortida de vídeo i àudio a través d'un connector HDMI, sortida d’àudio mitjançant un connector mini-jack, un connector fast ethernet 10/100 i un parell de ports USB.

A més la Raspberry Pi disposa d’un connector amb 28 pins de propòsit general (GPIO) per tal de connectar-li perifèrics.

El punt clau de la Raspberry Pi és tota la comunitat de desenvolupadors que hi ha al darrere que fan que aquesta placa sigui un referent

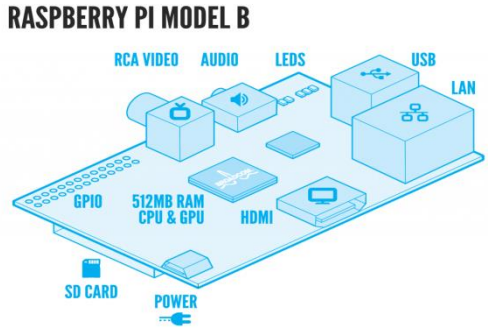


Figura 1. Components de la Raspberry Pi

Figura 2. Placa Raspberry Pi

Cubieboard

La **Cubieboard** és una placa microprocessadora de baix cost que inclou un SoC Allwinner A20, que conté un processador central (CPU) de doble nucli ARM Corex-A7 a 1 Ghz. També inclou un processador gràfic (GPU) de doble nucli Mali400 i 1 GB de memòria RAM. El disseny també inclou un connector SATA per tal de poder connectar un disc dur, una ranura per targetes microSD i una memòria internba NAND flash de 4GB. El preu d'una Cubiboard oscil·la sobre els 59$ segons el model.

A la placa ens trobem a més amb una sortida de vídeo i àudio a través d'un connector HDMI, un connector fast ethernet 10/100 i 4 ports USB.

Figura 3. Placa Cubieboard

1.2.2. Sistemes de reconeixement de veu

Avui en dia un òptim sistema de reconeixement de veu hauria de poder accedir a una gran quantitat de data que contingui mostres de diferents tonalitats de veu per tal de poder aproximar correctament la paraula reconeguda.

Els sistemes que incorporen comandes que poden ser reconegudes per qualsevol veu s'anomenen Speech Independenet (SI), els quals utilitzen una sèrie d'algoritmes de síntesis de veu per tal de reconèixer correctament la veu i transcriure-la a text. Aquests sistemes han de tenir una gran quantitat de mostres per tal de poder realitzar correctament la síntesis de veu.

Per altra banda, els sistemes que incorporen comandes que només depenen d'una única veu s'anomenen Speech Dependent (SD) i no requereixen de cap algoritme per tal de sintetitzar la veu ja que aquestes comandes poden ser "entrenades" per una única persona.

Per tal de duu a terme el reconeixement de veu hi ha les següents opcions:

- **Mitjançant un web service:** existeixen web services als quals enviant l'arxiu d'àudio amb la veu gravada executen els algoritmes de síntesis de veu pertinents i et retornen la cadena de caràcters corresponent a la frase o paraula dita.

  És un bon sistema ja que les comandes enregistrades seran Speech Independent ja que el web service per darrere ja s'encarrega d'executar els algoritmes que siguin necessaris i accedir a les mostres de veu necessàries per tal de detectar correctament el text pronunciat, però requereix que el nostre sistema disposi d'una connexió a internet.

  Existeix per exemple la **Google Speech API** amb la que podríem aconseguir el nostre propòsit.

- **Mitjançant un mòdul hardware amb comandes predefinides:** existeixen mòduls de reconeixement de veu que ja incorporen una petita base de dades de comandes SI i en els quals també es poden gravar noves comandes SD. La comunicació amb el mòdul es realitza generalment mitjançant una interfície sèrie UART.

  És un bon sistema si no es disposa de connexió a internet.

  Existeix per exemple el mòdul **EasyVR** de la marca Veear.

Figura 4. Mòdul de reconeixement de veu EasyVR 1.0

1.3. Objectius

En el projecte s'assoliran els següents objectius:

- Reconeixement de comandes de veu per tal d'identificar el pis i la porta de l'edifici al que es vol accedir mitjançant un mòdul hardware de reconeixement de veu. La comunicació d'aquest amb la placa microprocessadora es realitzarà mitjançant una interfície UART.

- Connectar correctament un polsador i un LED als pins de propòsit general de la placa microprocessadora.

- Capturar l'àudio provinent d'un micròfon connectat a la placa.

- Capturar imatges provinents d'una càmera connectada a la placa.

- Ser capaç d'enviar en temps real àudio i vídeo a un ordinador o a una altra placa microprocessadora.

- Poder visualitzar i reproduir correctament l'àudio i vídeo provinents de la placa microprocessadora que farà de mòdul extern.

- Creació d'una xarxa sense fils ad-hoc mitjançant un mòdul wi-fi connectat a la placa micropropcessadora per tal de poder connectar les dues entitats (mòdul extern i mòdul intern) i poder enviar l'àudio i el vídeo a través.

1.4. Esquema general

1.4.1. Esquema hardware

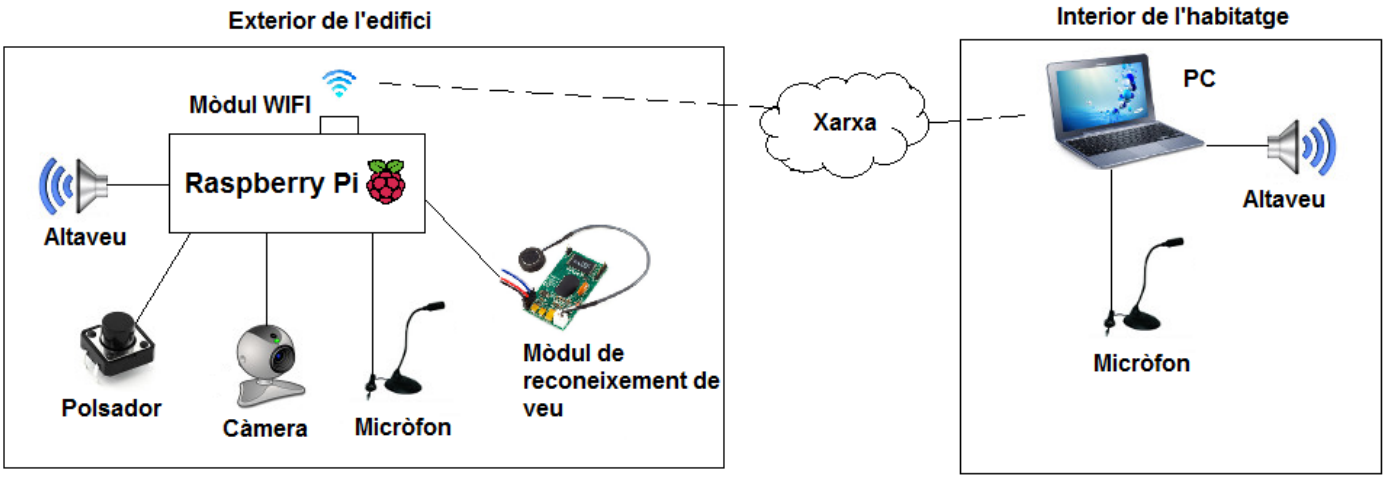


Figura 5. Esquema hardware del projecte

1.4.1. Esquema software

Raspberry Pi

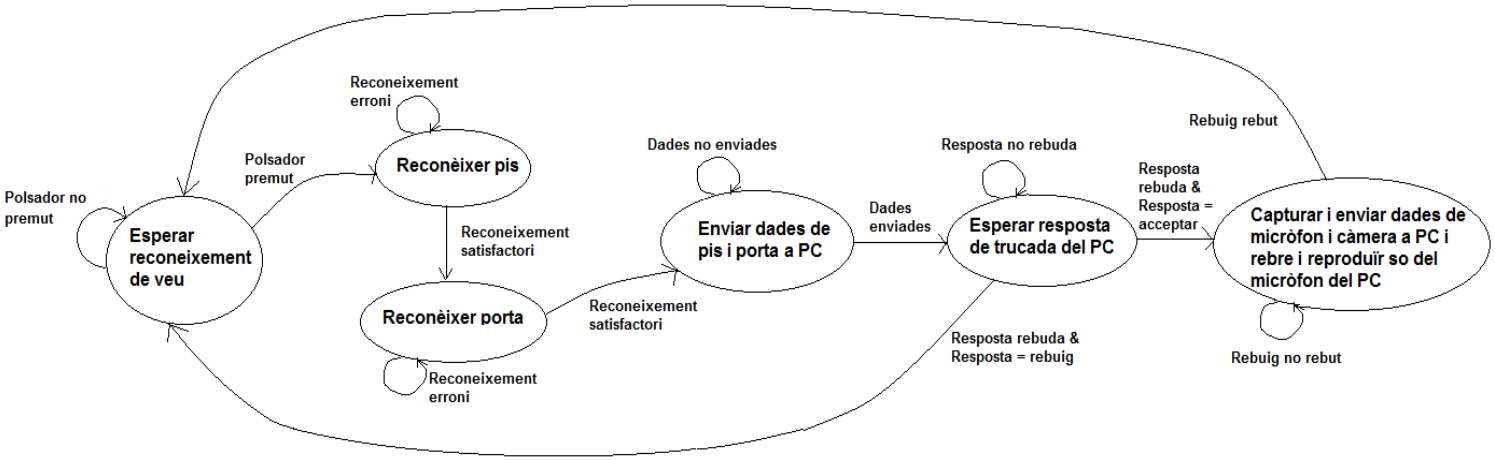


Figura 6. Graf d'estats del programa de la Raspberry Pi

PC

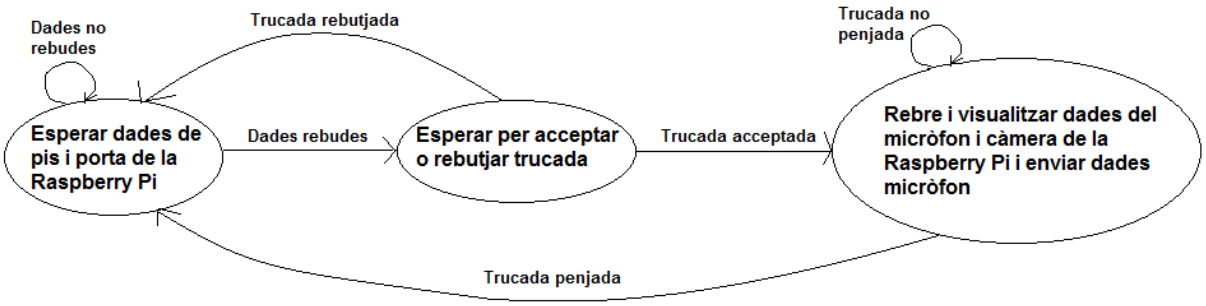


Figura 7. Graf d'estats del programa de PC

1.5. Selecció de components

Placa microprocessadora

Hi ha moltes plaques microprocessadores al mercat, i he escollit la **Raspberry Pi** per desenvolupar el projecte per les següents raons:

- És una placa amb molt bones prestacions hardware, té un processadora ARM que pot córrer fins a 1 Ghz de freqüència, té pins de propòsit general als quals poder connectar el mòdul de reconeixement de veu i el polsador i té dos ports USB per tal de connectar altres perifèrics.

- Té un cost molt reduït comparat amb altres solucions hardware (tan sols 35€).

- Té una gran comunitat de desenvolupadors al darrere i per tant una gran quantitat de projectes ja desenvolupats per a aquesta placa.

- Té unes reduïdes dimensions (85.60mm x 56mm x 21mm) cosa que la fa ideal per al projecte ja que així aconseguim que el mòdul extern sigui compacte.

- Se li poden instal·lar multitud de distribucions de SO basades en Linux adaptades per processadors ARM. En aquest cas hem escollit el SO Raspbian basat en Debian.

Càmera

Al principi es va escollir utilitzar una càmera web connectada per USB mitjançant la llibreria OpenCV. Per tal de testejar la viabilitat de l'elecció d'aquesta càmera, vaig fer un petit programa que captés les imatges de la càmera mitjançant aquesta llibreria, les mostrés en l'entorn local de la Raspberry Pi i mesurés els fotogrames per segon (FPS) que podria oferir la càmera. Els resultats que es van obtenir per a diferents resolucions d'imatge són els següents:

Resolució	1920x1080	1024x768	640x480	320x240
FPS	1	4.7	4,3	7,3

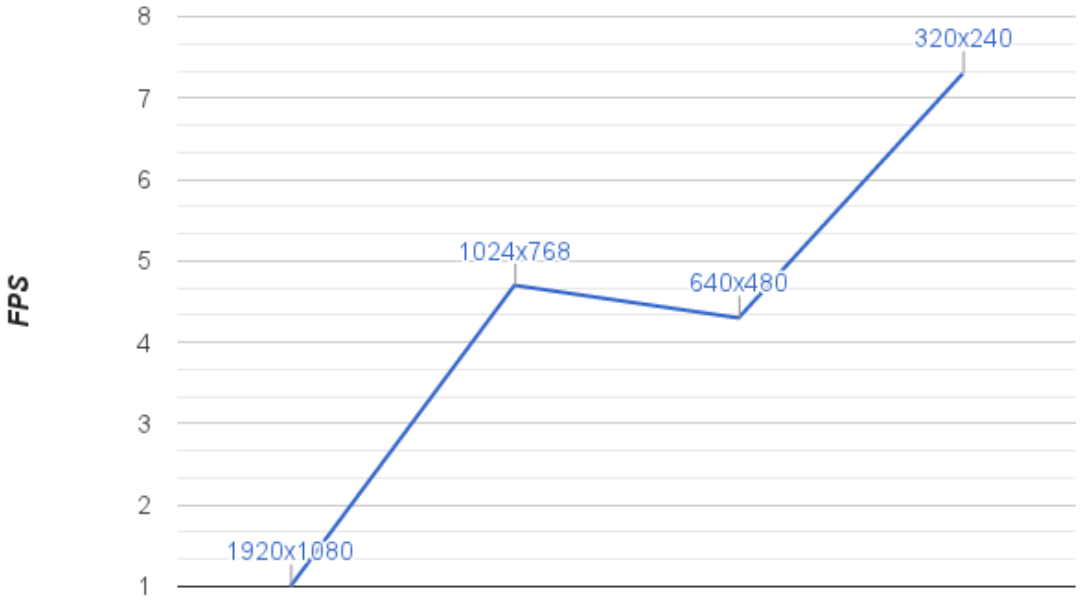


Figura 8. Gràfica dels fotogrames per segon d'una webcam usb a diferents resolucions

Com es pot observar a la gràfica si la resolució és menor dóna un framerate més alt, però tot i així, el framerate més alt que hem obtingut (7,3 FPS) no és òptim ja que el moviment apreciat no és gaire fluid (he d'esmentar que per a que l'ull humà noti una sensació de moviment fluid el framerate hauria de ser d'unes 24 imatges per segon).

Veient els resultats, l'opció de capturar les imatges d'una càmera web per usb amb OpenCV no és viable entre altres coses per la poca potencia que ofereixen els USBs de la Raspberry Pi, així que per poder obtenir un framerate més òptim he optat per comprar la **Raspberry Cam**, una càmera dissenyada especialment per la Raspberry Pi i la qual es connecta a aquesta mitjançant un port especial integrat en placa.

Figura 9. Raspberry Cam connectada a la RAspberry Pi

Tot seguit es mostra una taula comparativa del framerate d'aquesta càmera segons diferents resolucions per tal de justificar l'elecció:

Mode RGB:

Resolució	1280x960	640x480	320x240
FPS	28	29.29	29.24

Micròfon

Per tal d'enregistrar la veu s'utilitzarà un micròfon USB de peu per la Raspberry Pi i per al PC (en aquest cas s'utilitzarà un portàtil) s'emprarà el micròfon integrat.

Figura 10. Micròfon amb peu USB

Mòdul intern (PC)

Per tal d'emular el mòdul intern que aniria instal·lat dins de l'habitatge s'utilitzarà un portàtil amb Windows, però per tal de portar a terme el projecte a producció es podria utilitzar una altra Raspberry Pi amb Debian a la qual li connectaríem una pantalla mitjançant el connector HDMI del que disposa.

Mòdul WIFI

Per tal de dotar a la Raspberry Pi de connectivitat sense fils i poder crear una xarxa ad-hoc utilitzarem el mòdul nano USB wifi Edimax EW-7811UN ja que és dels nano USB wifi més populars dintre de la comunitat de desenvolupadors i el que més recomanen per la Raspberry Pi. A més aquest mòdul es pot configurar en mode ad-hoc.

Figura 11. Mòdul nano USB wifi Edimax

Targeta SD

Per tal d'instal·lar i emmagatzemar el sistema operatiu Raspbian que hem esmentat en l'apartat d'objectius utilitzarem la targeta Sandisk Extreme Pro de 16 GB ja que té espai de sobres per tal d'emmagatzemar el sistema operatiu i a més té una velocitat de lectura/escriptura de fins a 95 MB/s i d'aquesta manera minimitzem la latència d'escriptura en disc.

Figura 12. Targeta SD Sandisk 16 GB

Mòdul de reconeixement de veu

Per tal de duu a terme el reconeixement de veu he optat pel mòdul Easy VR 1.0 de la marca Veear. És un mòdul de reconeixement de veu d'usos múltiples dissenyat per afegir fàcilment capacitats de reconeixement de veu a gairebé qualsevol aplicació.

El mòdul EasyVR 1.0 es pot utilitzar amb qualsevol host amb una interfície UART alimentat a 3.3V - 5V. En el nostre cas configurarem un parell de pins de la Raspberry Pi per tal de que funcionin com a pins de transmissió i recepció de la interfície UART.

Al no disposar de connexió a internet, aquest mòdul és la millor opció per el projecte ja que tenim la possibilitat de gravar les nostres pròpies comandes de veu (aquestes comandes seran Speech Dependent) i a més disposa de diverses comandes Speech Independent que inclouen números cardinals en diferents idiomes.

Figura 13. Mòdul de reconeixement de veu EasyVR 1.0

2. Descripció del projecte

2.1. Desenvolupament per a la Raspberry Pi

2.1.1. Creació d'una xarxa ad-hoc

Com hem comentat abans per tal de interconnectar la Raspberry Pi que farà de mòdul intern i l'ordinador que farà de mòdul extern crearem una xarxa ad-hoc des des de la Raspberry Pi mitjançant el mòdul USB wifi Edimax.

Per duu a terme la tasca seguirem els següents passos:

- **Configuració del driver:**

  En el sistema operatiu Raspbian ve instal·lat amb un driver genèric Realtek 8192cu per tal de poder fer servir mòduls usb wifi. Per defecte el driver no requereix de cap configuració addicional, però el mòdul wifi té un mode de suspensió per tal d'estalviar energia en períodes llargs de no utilització. Com es pot donar el cas que no s'interactuï amb el mòdul extern en un període llarg de temps i no volem que la xarxa wifi caigui en cap moment haurem de configurar el mòdul per tal de desactivar el mode de suspensió.

  Per fer-ho hem de crear l'arxiu 8192cu.conf a la ruta /etc/modprobe.d/8192cu.conf amb el següent contingut:

  ```
  options 8192cu rtw_power_mgnt=0 rtw_enusbss=0
  ```

  Amb *rtw_power_mgnt= 0* desactivarem el mode d'estalvi d'energia i amb rtw_enusbss=0 desactivarem la auto suspensió del USB.

  D'aquesta manera cada vegada que s'engegui la RAspberry Pi, el driver accedirà a aquest arxiu i mitjançant aquestes opcions desactivarà el mode d'estalvi d'energia i així evitarem que la xarxa wifi es desconnecti.

- **Configuració del fitxer /etc/network/interfaces**

  Per tal de configurar correctament la interfície de xarxa wlan0 haurem d'afegir les següents línies en el fitxer /etc/network/interfases:

```
# Inicia la interfície quan el sistema s'engega
auto wlan0
 # Utilitzar una configuració manual d'IP per la interfície wlan0 i permetre desconnexió i connexió en calent.
allow-hotplug wlan0
iface wlan0 inet manual
```

- **Instal·lació i configuració del DHCP server**

  Per tal de que la Raspberry Pi pugui assignar una adreça IP als PCs que es connectin a la xarxa haurem d'instal·lar un DHCP server mitjançant les següents comandes:

```
sudo apt-get update
sudo apt-get install isc-dhcp-server
```

  Quan la instal·lació hagi acabar haurem de configurar alguns valors per defecte del DHCP server, per fer-ho obrirem l'arxiu /etc/default/isc-dhcp-server.

  L'únic canvi que hem de fer-li a l'arxiu és afegir el nom de la nostra interfície amb la opció INTERFACES per tal de que el DHCP server sàpiga en quina interfície ha d'atendre peticions.

```
INTERFACES="wlan0"
```

El següent pas és modificar l'arxiu de configuració del DHCP server /etc/dhcp/dhcpd.conf de la següent manera:

```
DHCPDARGS=wlan0; #arguments per al daemon DHCP (limitar el DHCP a la interfície wlan0)
default-lease-time 600;
max-lease-time 7200;

#Màscara de subxarxa
option subnet-mask 255.255.255.0;
#Adreça de broadcast
option broadcast-address 192.168.1.255;
#Nom del domini
option domain-name "RPi-network";
#Porta d'enllaç predeterminada de la xarxa
option routers 192.168.1.1;

subnet 192.168.1.0 netmask 255.255.255.0 {
   range 192.168.1.3 192.168.1.40; #Rang IP a oferir
}

#Assignació estàtica d'IP
host PC {
   hardware ethernet 11:aa:22:bb:33:cc;
   fixed-address 192.168.1.3;
}
```

Com es pot veure al final de l'arxiu, hem configurat una assignació estàtic d'IP per tal d'assegurar-nos que al PC se li assigna la direcció IP que nosaltres volem i no una automàtica.

- **Script per la creació de la xarxa**

  Per tal de que quan s'engegui la Raspberry Pi es creï una xarxa ad-hoc haurem de crear un bash script i configurar-lo per a que s'executi al inicialitzar-se el sistema.

  L'script té el següent contingut:

```
#!/bin/bash
echo "Creant la xarxa ad-hoc"
#Desactivem la interfície
ifconfig wlan0 down
#Configurem la interfície en mode ad-hoc
iwconfig wlan0 mode ad-hoc
#Li donem a la xarxa un SSID (Service Set Identifier)
iwconfig wlan0 essid RPi-network
#Configurem la xarxa amb una clau WEP
iwconfig wlan0 key aaaaa11111
#Assignem una IP a la interfície
ifconfig wlan0 192.168.1.2 netmask 255.255.255.0 up
#Executem el DHCP server en la interfície wlan0
/usr/sbin/dhcpd wlan0
echo "Xarxa ad-hoc creada"
```

  Per tal de que s'executi al carregar el sistema mourem l'script a la carpeta /etc/init.d. Després li afegirem el següent contingut al principi de l'arxiu:

```
### BEGIN INIT INFO
# Provides:          adhocScript
# Required-Start:    $syslog
# Required-Stop:     $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Configuració xarxa ad-hoc
# Description:       Configuració xarxa ad-hoc
### END INIT INFO
```

Els paràmetres indicats en els comentaris signifiquen el següent:

Provides:
Normalment s'escriu el nom de script en aquest camp (sense l'extensió .sh si el nom del fitxer la té) i, per tant, altres scripts d'inici que requereixin d'aquest s'han d'iniciar en una etapa posterior, però en algun cas excepcional també es poden indicar el nom del servei (s) que l'script substitueix.

Required-Start:
Defineix les dependències que han d'estar executades abans d'executar l'script. Si no s'especifica significa que aquest script es pot iniciar immediatament després de l'arrencada, sense muntar sistemes de fitxers locals, ni registre del sistema, etc.

Required-Stop:
Defineix les dependències que s'han d'aturar després de que aquest script s'aturi.

Default-Start:
Default-Stop:
Defineix els run-levels on l'script es pot iniciar (o parar).

Short-Description:
Description:
Descripció curta i llarga de l'script.

Finalment per tal de que es creïn els soft links en elsa directoris de run-level corresponents executarem la comanda:

```
sudo update-rc.d adhocScript defaults
```

2.1.2. Transmissió de dades en temps real

2.1.2.1. El protocol RTSP (capa d'aplicació)

El protocol de streaming en temps real (RTSP) és un protocol a nivell d'aplicació per a l'enviament de dades amb propietats de temps real. El seu paper és el de proporcionar control remot a l'aplicació en temps real , la codificació i enviament de dades es realitza separadament, normalment mitjançant el protocol RTP.

Proporciona un entorn per a l'enviament de dades de temps real sota demanda, com ho són l'àudio i el vídeo. Les fonts de dades poden incloure tant dades en directe, com emmagatzemades.
El servidor proporciona el contingut multimèdia, els clients demanen contínuament dades al servidor, i RTSP es comporta com un comandament a distància entre un client i el servidor , que permet:

- **Aconseguir dades del servidor.** El client li demana al servidor que configuri una sessió per enviar les dades sol · licitades.

- **Convidar un servidor a una conferència.**

- **Afegir dades a una presentació existent**. El client o el servidor poden notificar a l'altra part sobre dades que es troben disponibles.

RTSP intenta donar els mateixos serveis per a àudio i vídeo que HTTP fa per text i gràfics, i ha estat dissenyat intencionadament per a que tingui una sintaxi i operacions similars . Cada flux està identificat per una URL RTSP. Cada presentació i les seves propietats multimèdia queden recollides en un fitxer de descripció de la presentació, i aquest fitxer pot ser obtingut pels clients via HTTP, per email o qualsevol altre mitjà.

Però RTSP difereix d'HTTP en alguns aspectes: mentre que HTTP és un protocol sense estats, un servidor RTSP ha de mantenir els estats de les sessions per fer correspondre comandes i fluxos. I a més , HTTP és un protocol asimètric on el client fa peticions i el servidor respon, mentre que a RTSP tots dos poden fer peticions.

Els serveis i operacions suportades són les següents :

- **OPTIONS:** El client o el servidor li comuniquen a l'altra part quines opcions accepten.

- **DESCRIBE:** El client aconsegueix una descripció d'un contingut identificat per una URL RTSP.

- **ANNOUNCE:** Actualitza la descripció en temps real.

- **SETUP:** El client li pregunta al servidor on aconseguir les dades.

- **PLAY:** El client li demana al servidor que comenci a manar els fluxos configurats en SETUP .

- **PAUSE:** El client deté l'enviament sense alliberar els recursos al servidor.

- **TEARDOWN:** El client sol · licita al servidor que aturi l'enviament del flux especificat i alliberi tots els recursos associats a ell.

- **GET_PARAMETER:** Aconsegueix el valor d'un paràmetre d'una presentació o flux.

- **SET_PARAMETER:** Estableix el valor d'un paràmetre d'una presentació o flux.

- **REDIRECT:** El servidor informa al client que s'ha de connectar al servidor indicat en la capçalera.

- **RECORD:** El client comença a gravar dades de la presentació.

Les peticions RTSP són enviades normalment per un canal independent al canal de les dades . Aquests últims poden ser transmesos per un altre tipus de connexió.

2.1.2.2. Els protocols RTP i RTCP (capa de transport)

El protocol de transport en temps real (RTP) proporciona funcions per a xarxes d'extrem a extrem adequades per a aplicacions que transmeten dades en temps real, com àudio, vídeo, o dades provinents d'una simulació, sobre xarxes unicast o multicast .
El transport de dades és acompanyat per un protocol de control (RTCP) que permet monitoritzar l'enviament de dades de forma escalable en xarxes multicast.

RTP i RTCP han estat dissenyats per a ser independents de les capes de transport i xarxa sobre les que funcionin. RTP corre normalment sobre UDP per fer ús dels seus serveis de multiplexació i detecció d'errors. No obstant això, RTP pot ser usat sobre altres protocols de transport i de xarxa, fins i tot IPv6 .

RTP suporta transferència de dades a múltiples destinacions usant distribució multicast si el proporciona la xarxa sobre la que s'usa. Aquest protocol per si sol, no proporciona cap mecanisme per assegurar un enviament a temps ni garanteix qualitat de servei; delega en el protocol RVSP per a la reserva de recursos i proveir de qualitat de servei.

Això no garanteix que arribin les dades ni que arribin en ordre. Els números de seqüència inclosos en RTP permeten al receptor reconstruir la seqüència de paquets de l'emissor. RTP ha estat dissenyat principalment per a multimèdia, però no està limitat a aquesta aplicació, i també es pot aplicar a emmagatzematge de dades contínues, simulacions distribuïdes i interactives o aplicacions de control .

El protocol de control en temps real (RTCP) és el protocol de control dissenyat per treballar en conjunció amb RTP. En una sessió RTP, els participants envien periòdicament paquets RTCP amb informació referent a la qualitat de les dades rebudes.

Aquests paquets de control ens ofereixen els següents serveis :

- **Monitorització de la qualitat de servei i control de la congestió.** És la funció principal de RTCP, proporcionant la qualitat de la distribució de les dades. És útil tant per emissors, com receptors, com a terceres parts. L'emissor pot ajustar la transmissió en funció de l'informe rebut.

- **Identificació de la font.** En els paquets RTP els emissors són identificats per un nombre de 32 bits generat aleatòriament.

- **Sincronització entre fluxos .** Un paquet RTCP conté una indicació de temps real i el corresponent timestamp de RTP. Això pot usar-se per sincronització entre fluxos multimèdia com sincronització entre àudio i vídeo .

- **Informació de control escalable**. Els paquets RTCP són enviats periòdicament pels participants i quan el nombre d'aquests s'incrementa és necessari un equilibri entre tenir informació de control actualitzada i limitar el trànsit de control. RTP limita el trànsit de control a un màxim del 5 % de tot el trànsit de la sessió.

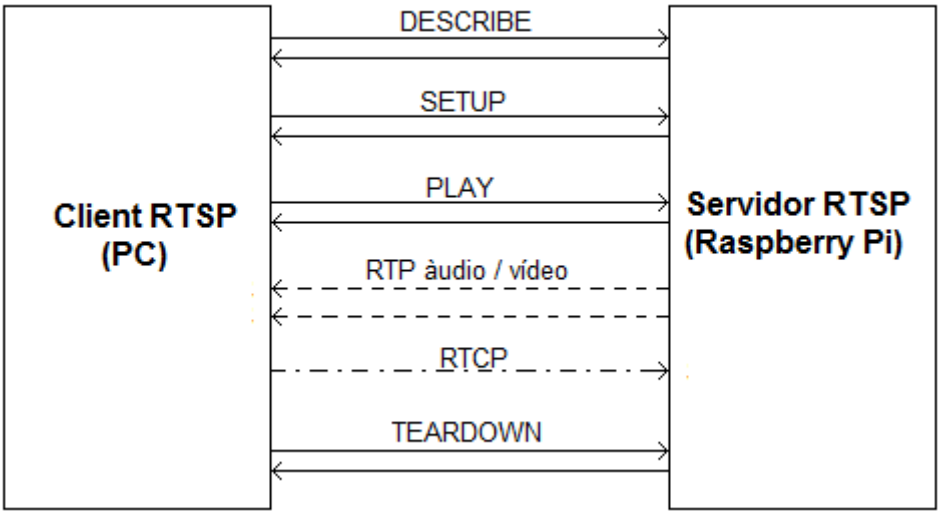


Figura 14. Esquema de comunicació RTSP entre servidor i client

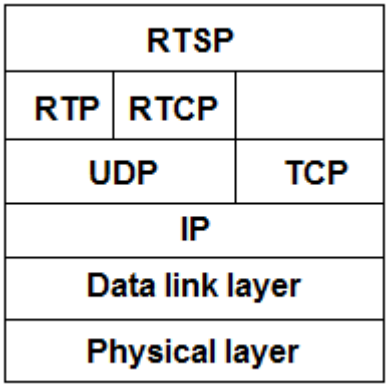


Figura 15. Torre de protocols RTSP

2.1.2.3. Implementació: ffserver i ffmpeg

Per tal d'implementar el servidor RTSP a la Raspberry Pi hem escollit utilitzar el programa ffserver.

Ffserver és un servidor d'streaming d'àudio i vídeo . És compatible amb diverses transmissions en viu, streaming dels arxius i desplaçament de temps dintre de les propies transmissions. Es pot buscar a una posició determinada de cada senyal en viu, sempre i quan s'especifiqui un tany d'arxiu prou gran.

Ffserver es configura mitjançant un fitxer de configuració , que es llegeix en l'inici . Si no s'especifica explícitament, es llegirà de '/etc/ffserver.conf '.

Ffserver rep arxius pregravats o fluxos FFM des d'algun procés ffmpeg com a entrada, i després els transmet mitjançant RTP/RTSP/HTTP. FFM i FFM2 són formats utilitzats per ffserver. Ells permeten emmagatzemar una àmplia varietat de fluxos de vídeo i d'àudio i opcions de codificació, i poden emmagatzemar un segment de temps en moviment d'una pel·lícula infinita o una pel·lícula sencera.

El servidor ffserver escoltarà en un port en concret especificat a l'arxiu de configuració. Es pot iniciar una o més instàncies de ffmpeg i enviar un o més streams FFM fins al port on ffserver està esperant per rebre'ls. Alternativament, es pot configurar per a que ffserver llanci diferents processos ffmpeg quan s'inicia.

Els fluxos d'entrada s'anomenen feeds, i cadascun s'especifica per una secció <*Feed*> a l'arxiu de configuració .

Per a cada feed es pot tenir diferents streams de sortida en diversos formats, cadascun especificat per una secció *<Stream>* a l'arxiu de configuració.

Ffserver treballa enviant streams codificats per el programa **ffmpeg** o enviant arxius pregravats que es llegeixen des del disc.

Precisament, ffserver actua com un servidor HTTP, acceptant sol·licituds POST de ffmpeg per tal de poder adquirir les dades de l'stream a publicar, i que atén a clients RTSP (com és el nostre cas) amb el contingut multimèdia .

Un feed és un stream FFM creat per ffmpeg, I com hem comentat abans s'envia a un port on ffserver està escoltant.

Cada feed s'identifica per un nom únic, que correspon al nom del recurs publicat en ffserver, i està configurat per una secció *<Feed>* a l'arxiu de configuració.

La URL del feed que utilitzarà ffmpeg per transmetre les dades serà del tipus:

```
http://ffserver_ip_address:http_port/feed_name
```

on *ffserver_ip_address* és l'adreça IP de la RaspberryPi on està instal·lat ffserver, *http_port* és el número de port del servidor HTTP (configurat a través de l'opció *Port* de l'arxiu de configuració), i *feed_name* és el nom del feed corresponent definit al fitxer de configuració .

Cada feed s'associa a un arxiu que s'emmagatzema en el disc . Aquest fitxer emmagatzemat s'utilitza per permetre enviar dades gravades prèviament a un client tan ràpid com sigui possible quan s'afegeix nou contingut en temps real a l'stream.

Cada stream s'identifica per un nom únic, que correspon al nom del recurs servit per ffserver, i està configurat per una secció *<Stream>* a l'arxiu de configuració.

La URL RTSP per la qual el client (PC) podrà reproduïr el contigut és del tipus:

rtsp://ffserver_ip_address:rtsp_port/stream_name[options]

on *stream_name* és el nom de l'stream definit al fitxer de configuració, *options* és una llista de les opcions especificades després de la URL que afecta la forma en l'stream és servit per ffserver i *rtsp_port* és el port RTSP configuraten l'opció *RTSPPort* de l'arxiu de configuració.

Com en el nostre cas casa stream està associat a un feed, els paràmetres de codificació s'han de configurar en la configuració de l'stream. Aquests són enviats a ffmpeg a l'hora de configurar la codificació. Això permet a ffserver definir els paràmetres de codificació que utilitzaran els codificadors de ffmpeg.

Les comandes exactes que utilitzarem per tal de transmetre les dades de la càmera de la Raspberry Pi, l'àudio del micròfon connectat a la placa i el micròfon del PC són les següents:

Raspberry Cam:

ffmpeg -r 20 -s 320x240 -f v4l2 -i /dev/video0 -vcodec mpeg4 http://192.168.1.39:8090/raspicam.ffm

Micròfon placa:

ffmpeg -f alsa -ac 2 -ar 44100 -i plughw:1,0 -acodec libmp3lame http://192.168.1.39:8090/raspimicro.ffm

Micròfon PC:

ffmpeg -f dshow -ac 2 -ar 44100 -i audio="Microphone (Realtek High Definition Audio)" -acodec libmp3lame http://192.168.1.39:8090/pcmicro.ffm

Les opcions de les comades es defineixen a continuació:

-r: *Especificar els fotogrames per segon.*

-i: *Especificar el dispositiu d'entrada.*

-vcodec: *Especificar el códec de vídeo*

-acodec: *Especoficar el códec d'àudio.*

-ac: *Especificar el número de canals d'àudio.*

-ar: *Especificar la freqüènciade mostreig d'àudio (en Hz)*

-f: *Forçar format de fitxer de sortida o d'entrada. El format és normalment auto-detectat per als arxius d'entrada i endevinat per l'extensió d'arxiu per als arxius de sortida, de manera que no és necessària aquesta opció en la majoria dels casos.*

Nota: v4l2 *és el format per als dispositius de vídeo sota linux,* ***alsa*** *és el format per als dispositius d'àudio sota linux (en el nostre cas control·lats per el driver alsa) i* ***dshow*** *és el format per als dispositius d'entrada en màquines windows.*

Un esquema de com estan estructurats els feeds i streams en la implemetació és el següent:

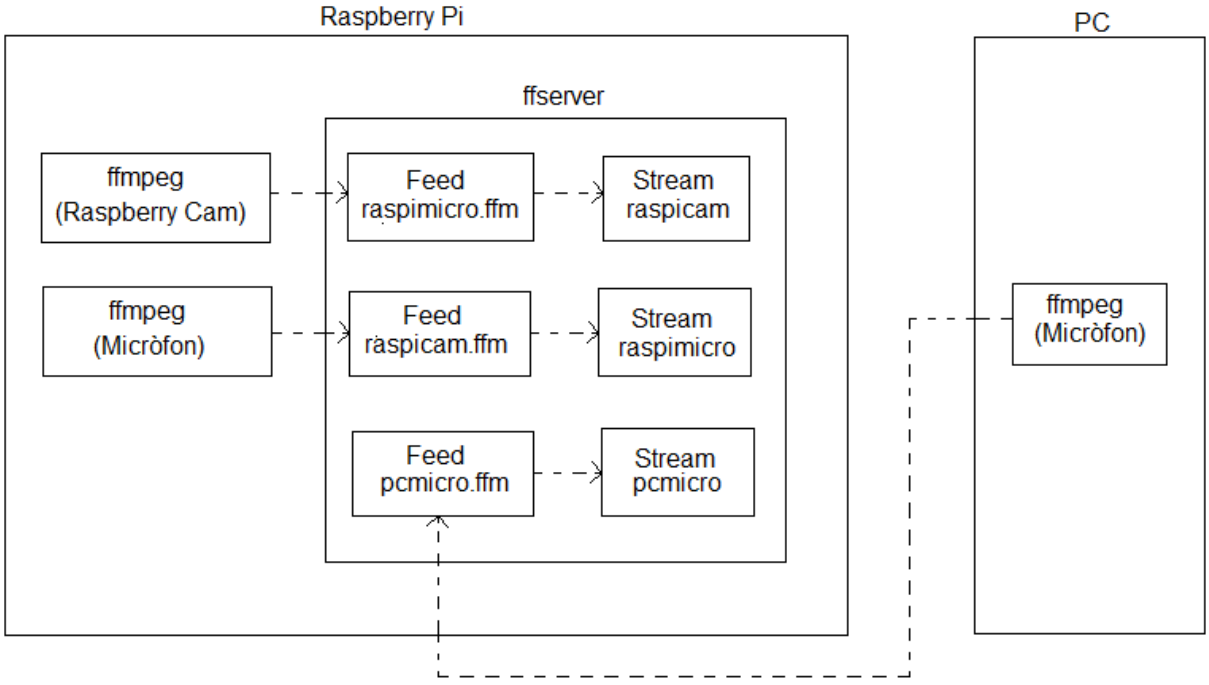


Figura 16. Estructura dels feeds i streams de ffserver

2.1.2.3.1. Arxiu de configuració

```
#Port del servidor HTTP on ffserver estarà escoltant
Port 8090
#Adreça IP del servidor HTTP on ffserver estarà escoltant
BindAddress 192.168.1.39
#Port del servidor RTSP on ffserver estarà escoltant
RTSPPort 5554
#Adreça IP del servidor HTTP on ffserver estarà escoltant
RTSPBindAddress 192.168.1.39
MaxHTTPConnections 2000
#Màxim de clients que sol·licitaran dades (només tindrem un PC connectat
a la vegada)
MaxClients 1
#Màxim ample de banda que el servidor podrà oferir
MaxBandwidth 9999999

<Feed raspicam.ffm>
      File /tmp/raspicam.ffm
      FileMaxSize 5M
        #Llista de control d'accés que indica les adreces IP que podran
        accedir al feed
        ACL allow 192.168.1.39
</Feed>

<Stream raspicam>
     Format rtp
     Feed raspicam.ffm
        VideoCodec mpeg4
     VideoFrameRate 20
        VideoBufferSize 80000
     VideoSize 320x240
        NoAudio
```

```
#Aquesta opció serveix per a que el reproductor que accedeixi a l'stream
sàpiga quins paràmetres utilitzar per reproduïr l'arxiu (utilitzarà els
paràmetres que s'indiquen dins de la secció Stream)
        AVOptionVideo flags +global_header
        ACL allow 192.168.1.33
</Stream>

<Feed raspimicro.ffm>
      File /tmp/raspimicro.ffm
      FileMaxSize 5M
      ACL allow 192.168.1.39
</Feed>

<Stream raspimicro>
      Format rtp
      Feed raspimicro.ffm
      AudioCodec libmp3lame
      AudioChannels 2
      AudioSampleRate 44100
      NoVideo
      AVOptionAudio flags +global_header
      ACL allow 192.168.1.33
</Stream>

<Feed pcmicro.ffm>
      File /tmp/pcmicro.ffm
      FileMaxSize 5M
      ACL allow 192.168.1.33
</Feed>
```

```
<Stream pcmicro>
    Format rtp
    Feed pcmicro.ffm
    AudioCodec libmp3lame
    AudioChannels 2
    AudioSampleRate 44100
    NoVideo
    AVOptionAudio flags +global_header
    ACL allow 192.168.1.33
</Stream>
```

Nota: l'adreça IP 192.168.1.39 correspon a l'adreça IP de la Raspberry Pi i l'adreça IP 192.168.1.33 correspon a la del PC

En la següent captura es mostra el log del ffserver on es poden observar algunes de les peticions del protocol RTSP:

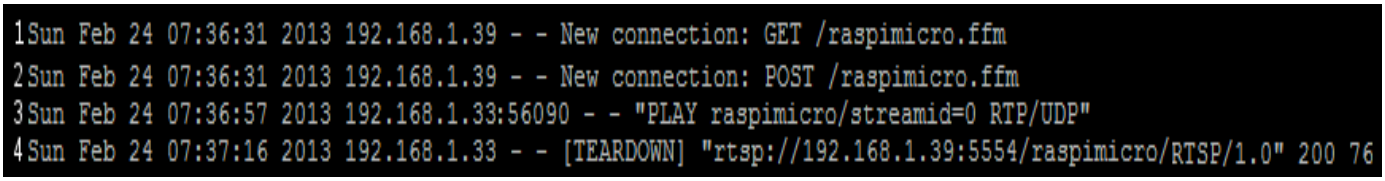


Figura 17. Captura de pantalla del log del programa ffserver

1. La Raspberry Pi ha realitzar una petició GET del feed corresponent al micròfon connectat a la placa.
2. La Raspberry Pi ha realitzar una petició POST del feed corresponent al micròfon connectat a la placa.
3. El PC ha realitzat una petició PLAY mitjançant el protocol RTSP
4. El PC ha realitzat una petició TEARDOWN mitjançant el protocol RTSP

2.1.3. Reconeixement de veu

2.1.3.1. Connexió del mòdul i el polsador als pins GPIO de la Raspberry Pi

Per tal de poder connectar el mòdul de reconeixement de veu EasyVR i el polsador haurem de localitzar els pins corresponents a la interfície UART i buscar un pin lliure per connectar el polsador.

La següent imatge mostra quins GPIO corresponen a quin pin del connector de la placa:

Figura 18. Esquema dels pins GPIO de la Raspberry Pi

Com podem observar els **GPIO 14 i 15** corresponen als pins TXD (transmissió) i RXD (recepció) de la interfície UART. Alternativament es podrien configurar aquests pins per a que realitzessin alguna altra funció, però per defecte estan configurats per realitzar les funcions de la interfície UART.

Llavors connectarem el mòdul a la placa de la següent forma:

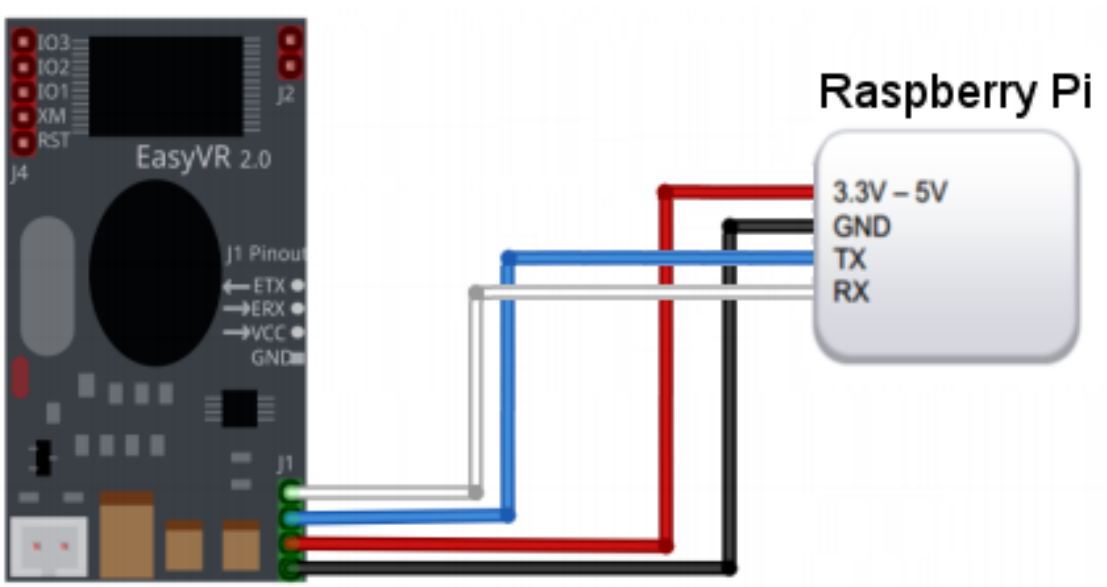


Figura 19. Connexió del mòdul EasyVR a la placa Raspberry Pi

Com podem observar el pin ETX es connecta amb el pin RX de la placa i el pin ERX amb el pin TX de la placa.

Per connectar el polsador utilitzarem el pin corresponent al GPIO 18 (just al costat del pin RXD) que el configurarem com a entrada i el connectarem de la següent forma mitjançant una resistència de pull-down:

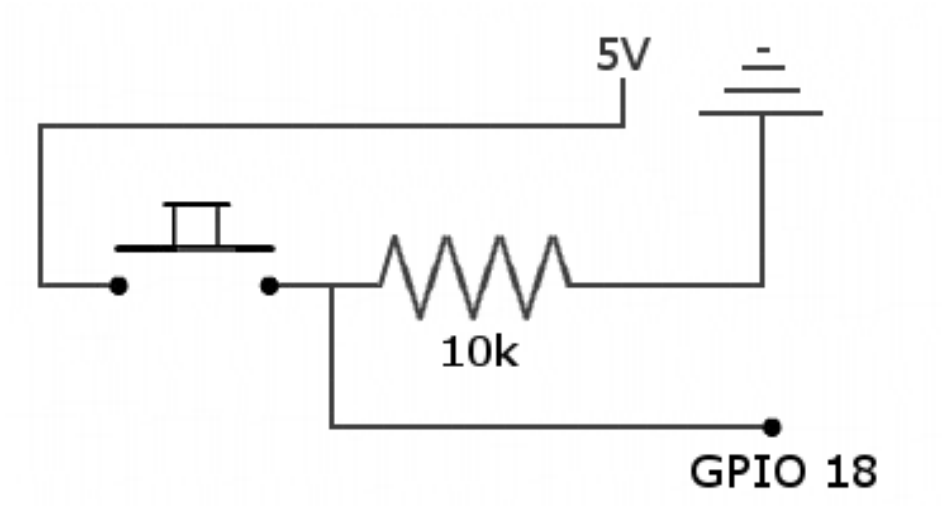


Figura 20. Connexió del polsador a un GPIO

2.1.3.2. Configuració de la interfície UART

Abans de res, per poder usar els pins de la UART de la Raspberry Pi hem de desactivar la funció principal per la que estan designats. Per defecte aquests pins s'utilitzen per oferir informació de debugging (el SO utilitza els pins per dipositar informació).

Per tal de fer això hem d'editar els arxius "/boot/cmdline.txt" i "/etc/inittab".

Haurem d'eliminar els següents paràmetres de configuració del fitxer "/boot/cmdline.txt":

```
console=ttyAMA0,115200" and "kgdboc=ttyAMA0,115200"
```

I finalment comentarem l'última línia en el fitxer "/etc/inittab". Posarem un '#' abans de :

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

2.1.3.3. Protocol de comunicació

La configuració inicial a l'inici és de 9600 bauds, 8 bits de dades, sense paritat i 1 bit de parada. La velocitat de transmissió pot ser canviada per operar en el rang entre 9600-115200 bauds .

El protocol de comunicació només utilitza caràcters ASCII imprimibles, que poden dividir-se en dos grups principals :

- Caràcters de comandament i d'estat, respectivament, en les línies de transmissió i recepció, escollits amb lletres minúscules.

- Arguments de comandament o detalls d'estat, de nou en les línies de transmissió i recepció, que abasten la gamma de les lletres majúscules.

Cada ordre enviada a la línia TX, amb zero o més bytes d'arguments addicionals, rep una resposta a la línia de RX en la forma d'un byte d'estat seguit de zero o més arguments.

Des de que la interfície sèrie EasyVR també pot ser basada en software, un retard molt curt podria ser necessari abans de la transmissió d'un nou caràcter al mòdul, sobretot si el host és molt ràpid, per permetre així que el mòdul EasyVR pugui tornar a escoltar la recepció d'un nou caràcter.

La comunicació és host-driven i cada byte de la resposta a una comanda ha de ser reconegut pel host per rebre dades addicionals d'estat, amb el caràcter espai. La resposta s'avorta si qualsevol altre caràcter es rep i llavors no hi ha la necessitat de llegir tots els bytes de resposta si no es requereix.

Combinacions no vàlides d'ordres o arguments són senyalats per un byte d'estat específic, que el host ha d'estar preparat per rebre si falla la comunicació. També un temps d'espera raonable s'ha d'utilitzar per recuperar-se de fallades inesperades.

Si el host no envia tots els arguments necessaris d'una comanda, la comanda és ignorada pel mòdul, sense previ avís, i l'amfitrió pot començar a enviar una altra comanda.

El mòdul passa automàticament al mode de baix consum després d'encendre's. Per iniciar la comunicació, s'ha d'enviar qualsevol caràcter per reactivar el mòdul.

2.1.3.3.1. Mapeig d'arguments

Els missatges de comandament o d'estat que s'envien a través de l'enllaç sèrie poden tenir un o més arguments numèrics en el rang de -1 a 31, que es codifiquen utilitzant majoritàriament caràcters de la sèrie de lletres majúscules. Aquestes són algunes constants útils per manejar arguments fàcilment:

ARG_MIN

'@' (40h)	Valor mínim d'argument (-1)

ARG_MAX

'`' (60h)	Valor màxim d'argument (+31)

ARG_ZERO

'A' (41h)	Valor d'argument zero (0)

ARG_ACK

' ' (20h)	Llegir més arguments d'estat

Tenir aquestes constants definides en el codi pot simplificar els controls de validesa i el procés de codificació / descodificació. Per exemple (en pseudo-codi):

```
#Codificar el valor 5
FIVE = 5 + ARG_ZERO
#Control de validesa
IF ARG < ARG_MIN OR ARG > ARG_MAX THEN ERROR
```

2.1.3.3.2. Tipus de comandes

En aquesta secció es descriu el format d'algunes de les comandes més importants acceptades pel mòdul. S'ha de tenir en compte que els arguments numèrics de les comandes s'assignen a lletres majúscules (com s'ha comentat en l'apartat anterior).

CMD_BREAK

'b' (62h)	Atura el reconeixement, entrenament o reproducció si n'hi ha o sinó no fa res.
Respostes esperades: STS_SUCCESS, STS_INTERR	

CMD_LANGUAGE

'l' (6Ch)	Indicar l'idioma de les comandes SI (Speech Independent)
[1]	Idioma: 0 = English 1 = Italian 2 = Japanese 3 = German 4 = Spanish 5 = French
Respostes esperades: STS_SUCCESS	

CMD_TIMEOUT

'o' (6Fh)	Especifica el timeout de reconeixement de veu
[1]	Timeout (-1 = per defecte, 0 = infinit, 1-31 = segons)
Respostes esperades: STS_SUCCESS	

CMD_RECOG_SI

'i' (69h)	Activar el reconeixement de comandes SI dintre d'un grup de paraules (word set)
[1]	Índex del grup de paraules (0-3)
Respostes esperades: : STS_SIMILAR, STS_TIMEOUT, STS_ERROR	

CMD_TRAIN_SD

't' (74h)	Entrenar una comanda SD (Speech Dependent) específica
[1]	Índex del grup (0 = trigger, 1-15 = generic, 16 = password)
[2]	Posició de la comanda (0-31)
Respostes esperades: STS_SUCCESS, STS_RESULT, STS_SIMILAR, STS_TIMEOUT, STS_ERROR	

CMD_GROUP_SD

'g' (67h)	Insertar una nova comanda SD
[1]	Índex del grup (0 = trigger, 1-15 = generic, 16 = password)
[2]	Posició (0-31)
Respostes esperades: : STS_SUCCESS, STS_OUT_OF_MEM	

CMD_UNGROUP_SD

'u' (75h)	Eliminar una comanda SD
[1]	Índex del grup (0 = trigger, 1-15 = generic, 16 = password)
[2]	Posició (0-31)
Respostes esperades: STS_SUCCESS	

CMD_ERASE_SD

'e' (65h)	Eliminar l'entrenament d'una comanda SD
[1]	Índex del grup (0 = trigger, 1-15 = generic, 16 = password)
[2]	Posició (0-31)
Respostes esperades: STS_SUCCESS	

CMD_RECOG_SD

'd' (64h)	Activar el reconeixement de comandes SD dintre d'un grup
[1]	Índex del grup (0 = trigger, 1-15 = generic, 16 = password)
Respostes esperades: STS_RESULT, STS_SIMILAR, STS_TIMEOUT, STS_ERROR	

2.1.3.3.3. Tipus de respostes

Com hem vist en l'anterior apartat quan s'envia una comanda al mòdul aquest respon amb un missatge d'estat. Alguns dels possibles missatges d'estat que pot retornar el mòdul són els següents:

STS_ERROR

'e' (65h)	Senyal d'error de reconeixement
[1-2]	Dos valors positius que formen un codi d'error de 8 bits (error = [1] * 16 + [2])
En resposta a: CMD_RECOG_SI, CMD_RECOG_SD, CMD_TRAIN_SD, CMD_PLAY_SX	

STS_TIMEOUT

't' (74h)	El timeout ha expirat
En resposta a: CMD_RECOG_SI, CMD_RECOG_SD, CMD_TRAIN_SD	

STS_SUCCESS

'o' (6Fh)	OK o no errors d'estat
En resposta a: CMD_BREAK, CMD_DELAY, CMD_BAUDRATE , CMD_TIMEOUT, CMD_KNOB, CMD_LEVEL, CMD_LANGUAGE, CMD_SLEEP, CMD_GROUP_SD, CMD_UNGROUP_SD, CMD_ERASE_SD, CMD_NAME_SD, CMD_RESETALL, CMD_QUERY_IO, CMD_PLAY_SX	

STS_RESULT

'r' (72h)	Comanda SD reconeguda o entrenament similar a una altra comanda SD
En resposta a: CMD_RECOG_SD, CMD_TRAIN_SD	

STS_SIMILAR

's' (73h)	Comanda SI reconeguda o entrenament similar a una altra comanda SI
[1]	Índex de paraula (0-31)
En resposta a: : CMD_RECOG_SI, CMD_RECOG_SD, CMD_TRAIN_SD	

STS_OUT_OF_MEM

'm' (6Dh)	Error de memòria (no hi ha suficient espai o taula de sons no present)
En resposta a: CMD_GROUP_SD, CMD_DUMP_SX	

2.1.3.4. Entrenament de comandes de veu

El mòdul EasyVR disposa d'un conjunt de grups de paraules (SI) en diferents idiomes. Dins d'aquests grups de paraules el que més ens interessaria seria el grup de números cardinals (un, dos, tres...), però per poder identificar un pis necessitaríem números ordinals (primer, segon, tercer....) i per identificar la porta necessitaríem lletres (A, B, C...), així que l'opció que ens queda serà la de gravar i entrenar les nostres pròpies comandes SD (Speech independent).

Per tal de comunicar-nos amb el mòdul mitjançant la interfície UART i poder gravar i entrenar les comandes utilitzarem la consola de python.

Abans d'utilitzar la consola haurem d'instal·lar la llibreria serial de python. Descarregarem l'última versió de http://sourceforge.net/projects/pyserial/files/pyserial/ i executarem l'script setup.py per instal·lar-la:

```
python setup.py install
```

Per començar, obrirem la consola de python (comanda *python*) i executarem les següents comandes per tal de connectar-nos a la interfície UART:

```
import serial
ser = serial.Serial("/dev/ttyAMA0")
```

Tot seguit executarem les següents comandes per treure al mòdul del mode d'estalvi d'energia fent les esperes oportunes manualment:

```
ser.write('b') #valdria qualsevol caràcter
(esperar...)
ser.read() #Si tot ha anat bé rebrem la resposta 'o' (STS_SUCCESS)
```

I finalment per entrenar una comanda SD executarem les següents comandes (ho farem repetidament per tal d'entrenar tantes comandes com siguin necessàries, en el nostre cas 9 comandes per als pisos i 4 per a les portes):

```
ser.write('g') # CMD_GROUP_SD (afegit una comanda SD)
ser.write('B') # en el grup 1
ser.write('A') # en la posició 0
ser.read() #si tot va bé rebrem la resposta 'o' (STS_SUCCESS)

ser.write('t') # CMD_TRAIN_SD (entrenar comanda SD)
ser.write('B') # del grup 1
ser.write('A') # de la posició 0

(pronunciar la comanda)

ser.read() # si tot va bé rebrem la resposta 'o' (STS_SUCCESS)
(realitzarem el procés d'entrenar la comanda dues vegades per tal
d'assegurar-nos que s'ha reconegut correctament)
```

Per comprovar que hem gravat correctament la comanda, executarem el reconeixement de veu per comprovar-ho amb les següents comandes:

```
ser.write('d') #CMD_RECOG_SD (activar el reconeixement de comandes SD)
ser.write('B') # en el grup 1

(pronunciar la comanda)

ser.read() # si tot va bé rebrem la resposta 'r' (STS_RESULT)

ser.write(' ') #escriurem el caràcter espai per tal de que el mòdul ens respongui amb la posició de la comanda reconeguda dins del grup indicat abans

ser.read() #si tot ha anat bé llegirem la posició de la comanda correcta
```

2.1.4. Programa C++

2.1.4.1. Llibreries utilitzades

Per tal de desenvolupar el projecte s'ha fet us de la llibreria *wiringPi* per tal de facilitar-nos l'accés al por sèrie de la placa (UART) des de codi C++ i la llibreria *bcm2835* que ens proveeix d'accés als GPIO i a altres funcions IO del xip Broadcom BCM 2835 de la Raspberry Pi.

2.1.4.1.1. wiringPi

Algunes de les funcions que proveeix la llibreria i que hem utilitzat en el projecte són les següents:

- **int serialOpen (char* dispositiu , int bauds) :**

  Aquesta funció obre i inicialitza el dispositiu sèrie i estableix la velocitat de transmissió. Estableix el port en mode "raw " un caràcter alhora i no hi ha traduccions), i estableix el temps d'espera de lectura a 10 segons. El valor de retorn és el descriptor de fitxer o -1 per qualsevol error, en aquest cas errno s'establirà segons el cas.

- **void serialClose (int fd) :**

  Tanca el dispositiu identificat pel descriptor de fitxer indicat.

- **void serialPutchar (int fd, unsigned char c) :**

  Envia un únic byte al dispositiu sèrie identificat pel descriptor de fitxer indicat.

- **void serialPuts (int fd, char* s) :**

  Envia la cadena de caràcters terminada en nul al dispositiu sèrie identificat pel descriptor de fitxer indicat.

- **void serialPrintf (int fd, char * missatge , ...) :**

  Emula la crida a sistema printf per al dispositiu sèrie .

- **int serialDataAvail (int fd) :**

  Retorna el nombre de caràcters disponibles per a la lectura, o -1 per a qualsevol condició d'error, en aquest cas errno s'establirà segons el cas.

- **int serialGetchar (int fd) :**

  Retorna el següent caràcter disponible al dispositiu sèrie. Aquesta crida es bloquejarà durant un màxim de 10 segons si no es disposa de dades (quan tornarà -1).

- **void serialFlush (int fd) :**

  Aquesta funció descarta totes les dades rebudes o en espera de ser enviades pel dispositiu donat.

2.1.4.1.2. bcm2835

Algunes de les funcions que proveeix la llibreria i que hem utilitzat en el projecte són les següents:

- **int bcm2835_init ( ) :**

  Inicialitza la llibreria obrint /dev/mem i obtenint punters a la memòria interna per els registres de dispositiu del xip BCM2835. S'ha de cridar abans de cridar a qualsevol altra funció de la llibreria. Retorna 1 en cas d'èxit i 0 en cas d'error.

- **bcm2835_gpio_fsel (uint8_t pin, uint8_t mode) :**

  Estableix la funció de selecció del registre per al pin donat, es pot configurar el pin com a entrada, sortida o una de les 6 funcions alternatives.

- **uint8_t bcm2835_gpio_lev (uint8_t pin) :**

  Llegeix el nivell actual del pin i torna HIGH o LOW. Funciona tant si el pin està configurat com entrada o com a sortida.

Constants utilitzades

Algunes de les constants que estan definides a la llibreria i que hem utilitzat són els següents:

- **HIGH = 0x01 :**
  Significa que un pin està HIGH, és a dir, 3.3V en el pin.

- **LOW = 0x00 :**

  Significa que un pin està LOW, és a dir, 0V en el pin.

- **BCM2835_GPIO_FSEL_INPT = 0b000 :**

  S'utilitza per configurar la direcció d'un pin com a entrada.

- **RPI_V2_GPIO_P1_12 = 18 :**

  Defineix el número de GPIO (18) corresponent al pin P1_12 del xip.

2.1.4.2. Mòduls

Aquí es definiran els mòduls (classes) que s'han programat així com les seves funcions per tal de desenvolupar el programa principal.

2.1.4.2.1. cEasyVR

Aquest mòdul utilitza la llibreria wiringPi i implementa les funcions necessàries per tal de poder enviar comandes i arguments al mòdul EasyVR de reconeixement de veu. També fa us de l'arxiu protocols.h que conté les definicions de constants corresponents a les comandes i estats del mòdul EasyVR abans definits.
L'arxiu de capçaleres on hem definit les variables i funcions del mòdul és el següent:

```
#define NUM_PISOS 9
#define NUM_PUERTAS 4
class cEasyVR {

        private:
                vector<string> pisos;
                vector<string> puertas;
                int fd;
                string pisoaux;

                char readResult();
                void sendCmd(char cmd);
                void sendArg(int arg);
                bool wakeUp();
        public:
                cEasyVR();
                ~cEasyVR();
                string reconocerPiso();
                string reconocerPuerta();
};
```

Seguidament es defineixen les variables i funcions del mòdul:

Variables

- **vector<string> pisos :**
  Vector on es guarden els noms dels pisos que hem gravat en el mòdul de reconeixement de veu EasyVR. També guardarem els paths dels arxius d'àudio que descriuen els pisos en qüestió.

- **vector<string> puertas :**
  Vector on es guarden els noms de les portes que hem gravat en el mòdul de reconeixement de veu EasyVR. També guardarem els paths dels arxius d'àudio que descriuen les portes en qüestió.

- **int fd :**
  Variable que guarda el descriptor de fitxer del port sèrie de la placa on hem connectat el mòdul.

- **string pisoaux :**
  Variable que emmagatzema temporalment el nom del pis que s'ha reconegut per tal de que quan es reconegui també la porta poder reproduir els sons del pis i la porta en qüestió.

Funcions

- **cEasyVR () :**
  Funció creadora de la classe. Es fan les inicialitzacions pertinents de les variables, s'activa el mòdul i s'obre el canal del port sèrie.

```
cEasyVR::cEasyVR(){
        pisos = vector<string>(2*NUM_PISOS);
        pisos[0] = "Primero";
        pisos[1] = "Segundo";
        pisos[2] = "Tercero";
        pisos[3] = "Cuarto";
```

```
        pisos[4] = "Quinto";
        pisos[5] = "Sexto";
        pisos[6] = "Séptimo";
        pisos[7] = "Octavo";
        pisos[8] = "Noveno";
        pisos[9] = "/home/pi/TFG/sounds/primero.wav";
        pisos[10] = "/home/pi/TFG/sounds/segundo.wav";
        pisos[11] = "/home/pi/TFG/sounds/tercero.wav";
        pisos[12] = "/home/pi/TFG/sounds/cuarto.wav";
        pisos[13] = "/home/pi/TFG/sounds/quinto.wav";
        pisos[14] = "/home/pi/TFG/sounds/sexto.wav";
        pisos[15] = "/home/pi/TFG/sounds/septimo.wav";
        pisos[16] = "/home/pi/TFG/sounds/octavo.wav";
        pisos[17] = "/home/pi/TFG/sounds/noveno.wav";

        puertas = vector<string>(2*NUM_PUERTAS);
        puertas[0] = "A";
        puertas[1] = "B";
        puertas[2] = "C";
        puertas[3] = "D";
        puertas[4] = "/home/pi/TFG/sounds/A.wav";
        puertas[5] = "/home/pi/TFG/sounds/B.wav";
        puertas[6] = "/home/pi/TFG/sounds/C.wav";
        puertas[7] = "/home/pi/TFG/sounds/D.wav";

        #S'obre el canal amb el dispositiu sèrie
        fd  = serialOpen("/dev/ttyAMA0",9600);

        #Es crida a aquesta funció per tal treure el mòdul de l'estat
        d'estalvi d'energia
        wakeUp();
}
```

- **void sendCmd (char cmd) :**
  Funció que agafa el caràcter corresponent a la comanda que es vol enviar al mòdul i s'envia pel port sèrie. Fa us de funcions de la llibreria wiringPi.

```
void cEasyVR::sendCmd(char cmd) {
        serialFlush(fd);
        serialPutchar(fd, cmd);
}
```

- **void sendArg(int arg) :**
  Funció que agafa l'argument numèric relacionat amb una comanda enviada prèviament i l'envia pel port sèrie.

```
void cEasyVR::sendArg(int arg) {
        serialPutchar(fd, ARG_ZERO + arg);

}
```

- **char readResult ( ) :**
  Funció que llegeix el caràcter de la resposta del mòdul de reconeixement de veu fruit d'una comanda enviada anteriorment. La crida és bloquejant, és a dir, que l'execució no continuarà fins que no es rebin dades.

```
char cEasyVR::readResult() {
        while(serialDataAvail(fd) < 0);
        char result = serialGetchar(fd);
        return result;
}
```

- **bool wakeUp ( ) :**
  Funció que envia un caràcter qualsevol al mòdul per tal de treure'l de l'estat d'estalvi d'energia i poder començar a utilitzar-lo. També es modifica el timeout de reconeixement de veu i es configura a 2 segons.

```
bool cEasyVR::wakeUp() {
        sendCmd(CMD_BREAK);
        char c = readResult();
        while(c != STS_SUCCESS) c = readResult();
        sendCmd(CMD_TIMEOUT);
        sendArg(2);
        c = readResult();
        while(c != STS_SUCCESS) c = readResult();
        return true;
}
```

- **string reconocerPiso ( ) :**
  Funció que activa el reconeixement de veu i retorna l'string corresponent al pis reconegut. Aquesta funció es crida des del programa principal.

```
string cEasyVR::reconocerPiso() {
        int pid = fork();
        if(pid == 0) {
           execl("/usr/bin/aplay", "/usr/bin/aplay",
                        "/home/pi/TFG/sounds/piso.wav",NULL);
        }
       int status;
       waitpid(pid, &status, WUNTRACED);
        sendCmd(CMD_RECOG_SD);
        sendArg(1);
        char result = readResult();
```

```
        while(result != STS_RESULT) {
                sendCmd(CMD_RECOG_SD);
                sendArg(1);
                result = readResult();
        }
        sendCmd(' ');
        int pos = readResult() - ARG_ZERO;
        string res =  pisos[pos];
        pisoaux = pisos[pos + NUM_PISOS];
        return res;
}
```

- **string reconocerPuerta ( ) :**
  Funció que activa el reconeixement de veu i retorna l'string corresponent a la porta reconeguda. Aquesta funció es crida des del programa principal.

```
string cEasyVR::reconocerPuerta() {
        int pid = fork();
        if(pid == 0) {
            execl("/usr/bin/aplay", "/usr/bin/aplay",
                        "/home/pi/TFG/sounds/puerta.wav",NULL);
        }
        int status;
        waitpid(pid, &status, WUNTRACED);
        sendCmd(CMD_RECOG_SD);
        sendArg(2);
        char result = readResult();
        while(result != STS_RESULT) {
              sendCmd(CMD_RECOG_SD);
              sendArg(2);
              result = readResult();
        }
        sendCmd(' ');
```

```
        int pos = readResult() - ARG_ZERO;
        string res = puertas[pos];
        string puertaaux = puertas[pos + NUM_PUERTAS];

         pid = fork();
         if(pid == 0) {
             execl("/usr/bin/aplay", "/usr/bin/aplay",
                        "/home/pi/TFG/sounds/llamando.wav",NULL);
         }
         waitpid(pid, &status, WUNTRACED);

         pid = fork();
         if(pid == 0) {
            execl("/usr/bin/aplay", "/usr/bin/aplay",
                        pisoaux.c_str(),NULL);
         }
         waitpid(pid, &status, WUNTRACED);

         pid = fork();
         if(pid == 0) {
            execl("/usr/bin/aplay", "/usr/bin/aplay",
                        puertaaux.c_str(),NULL);
         }
         waitpid(pid, &status, WUNTRACED);

         return res;
}
```

2.1.4.2.2. cSockets

Aquest mòdul fa ús d'una sèrie de llibreries del sistema i implementa les funcions necessàries per tal de crear un socket i connectar-s'hi.
L'arxiu de capçaleres on hem definit les variables i funcions del mòdul és el següent:

```
class cSockets {
        public:
           cSockets();
           ~cSockets();
           void createSocket(string ip, int port);
           int  connectSocket();

        private:
           sockaddr_in addr;
           int sockfd;
};
```

Seguidament es defineixen les variables i funcions del mòdul:

Variables

- **sockaddr_in addr :**
  Variable que enmagatzema una estructura de dades que conté informació sobre el socket (tipus de socket, ip del servidor, port...)

- **int sockfd :**
  Variable que emmagatzema el descriptor de fitxer per tal de poder llegir i escriure al socket.

Funcions

- **void createSocket(string ip, int port) :**
  Funció que crea un socket amb la ip i port passats com a paràmetre.

```
void cSockets::createSocket(string  ip, int port) {
      #Inicialitza l'estructura de dades
     bzero(&addr, sizeof(addr));
     #Emplena l'estructura anterior amb les dades d'IP i port
     addr.sin_family = AF_INET;
     addr.sin_addr.s_addr = inet_addr(ip.c_str());
     addr.sin_port = htons(port);
    #Obre un socket TCP amb les dades anteriors
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     #Aquesta crida configura el descriptor de fitxer per tal de que
     les lectures que es facin siguin bloquejants
     ioctl(sockfd, FIONBIO, 0);
}
```

- **connectSocket ( ) :**
  Funció que es connecta al socket creat amb la funció createSocket. La crida és bloquejant, és a dir, que l'execució del programa no continuarà fins que no es connecti al socket. Retorna el descriptor de fitxer.

```
int  cSockets::connectSocket() {
        while(connect(sockfd, (struct sockaddr *) &addr, sizeof(addr))
        < 0) {
                cout << "Error al connectar-se al socket" << endl;
        }
        cout << "Connexió establerta correctament" << endl;

        return sockfd;
}
```

2.1.4.3. Programa principal

El programa principal que fa us del diferents mòduls i llibreries abans descrites és el següent:

```
int main() {
        cEasyVR* easyvr = new cEasyVR();
        cSockets* sockets = new cSockets();

        #Inicialització de la llibreria bcm2835
        if (!bcm2835_init()) return 1;
        #Configuració del pin del polsador com a entrada
        int pin = RPI_V2_GPIO_P1_12;
        bcm2835_gpio_fsel(pin, BCM2835_GPIO_FSEL_INPT);
        bool data;

        int ffserver,streamAudio, streamVideo, pcAudio;
        char *const environmentList[] ={"LD_LIBRARY_PATH=/usr/local/lib",
        "LD_PRELOAD=/usr/lib/uv4l/uv4lext/armv6l/libuv4lext.so", NULL};

        while(1) {
                #Posar en marxa el procés ffserver
                ffserver = fork();

                 if(ffserver == 0) {
                        execle("/usr/bin/nice","/usr/bin/nice","-n","-5",
                                "/usr/local/bin/ffserver", "-d", "-f",
                                "/etc/ffserver.conf", "&", NULL, environmentList);
                 }

                #Crea el socket i es connecta
                sockets -> createSocket("192.168.1.33", 1500);
                int fd =  sockets -> connectSocket();
```

```
#Llegeix el valor del pin i eentra en un bucle fins que el pin no sigui HIGH
data = bcm2835_gpio_lev(pin);
while(!data)  data = bcm2835_gpio_lev(pin);

#Crida al mòdul cEasyVR per tal de reconèixer el pis
string result = easyvr->reconocerPiso();

#Envia l'string amb el pis al PC i un caràcter separador ($)
write(fd, result.c_str(), result.size());
write(fd,"$",1);


#Crida al mòdul cEasyVR per tal de reconèixer la porta
result = easyvr->reconocerPuerta();

#Envia l'string amb la porta al PC i un caràcter separador ($)
write(fd, result.c_str(), result.size());
write(fd,"$",1);

char c;

#Posa en marxa el procés ffmpeg que captura les imatges de la càmera de la Raspberry Pi i les envia al servidor RTSP (ffserver).
streamVideo = fork();
if(streamVideo == 0) {
        execle("/usr/bin/nice", "/usr/bin/nice", "-n" , "-20",
                "/usr/local/bin/ffmpeg","-r", "20", "-s",
                "320x240", "-f", "v4l2", "-i", "/dev/video0", "-
                vcodec", "mpeg4",
                "http://192.168.1.39:8090/raspicam.ffm", NULL,
                environmentList);
 }
```

```
#Posa en marxa el procés ffmpeg que captura l'àudio del
micròfon connectat a la Raspberry Pi i ho envia al servidor
RTSP  (ffserver).
streamAudio = fork();
if(streamAudio == 0) {
        execle("/usr/bin/nice","/usr/bin/nice", "-n", "-10",
                "/usr/local/bin/ffmpeg", "-f", "alsa", "-ac", "2", "-
                ar", "44100","-i", "plughw:1,0", "-acodec",
                "libmp3lame",
                "http://192.168.1.39:8090/raspimicro.ffm",
                NULL, environmentList);
 }

#Llegeix la resposta del PC i si es rep un '1' es continua
if(read(fd, &c, 1) > 0) {
        close(fd);
        if(c ==  '1') {
                #Torna a crear el socket i es connecta i envia un
                        ack
                sockets -> createSocket("192.168.1.33", 1500);
                fd =  sockets -> connectSocket();
                char* ack = "RUN";
                write(fd, ack, sizeof(ack));

                #Posa en marxa el programa mplayer per tal de
                        reproduir l'àudio del micròfon del PC
                pcAudio = fork();
                if(pcAudio == 0) {
                        execl("/usr/bin/nice", "/usr/bin/nice", "-n",
                        "-2", "/usr/bin/mplayer",
                        "rtsp://192.168.1.39:5554/pcaudio",
                        NULL);
                 }
```

```
                        #Espera la resposta del PC i si es rep un '0'
                                s'aturen els programes abans engegats
                        if(read(fd, &c, 1) > 0) {
                                if(c == '0')  {
                                        kill(pcAudio, SIGKILL);
                                        kill(streamAudio, SIGKILL);
                                        kill(streamVideo, SIGKILL);
                                        kill(ffserver, SIGKILL);
                                }
                                close(fd);
                        }
                }
                #si es rep un '0' s'aturen els programes abans engegats
                else {
                        kill(streamAudio, SIGKILL);
                        kill(streamVideo, SIGKILL);
                kill(ffserver, SIGKILL);
                }
        }
    }
  return 1;
}
```

2.2. Desenvolupament per al PC

2.2.1. Programa C#

El programa de PC el desenvoluparem en .NET windowsForms per tal de tenir interfície gràfica i amb el llenguatge C#.

2.2.1.1. Llibreries utilitzades

2.2.1.1.1. VLC.DotNet

Per tal de poder obtenir un control en la pantalla de la nostra aplicació que ens permeti reproduir i visualitzar un contingut multimèdia provinent d'una URL RTSP hem instal·lat la llibreria VLC.DotNet que proveeix una sèrie de controls per tal de reproduir àudio i video en una aplicació de Windows Forms, WPF o Silverlight. Fa us de les llibreries del VLC Media Player així que per a que funcioni hem de tenir instal·lat prèviament aquest programa de reproducció en el nostre PC.

Concretament utilitzarem el control VlcControl de la llibreria que ens proveeix d'una pantalla per tal de reproduir el vídeo i l'àudio. Com nosaltres disposem de dos fluxos (streams) de dades de reproducció provinents de la Raspberry Pi (un stream per a l'àudio i un altre per al vídeo) utilitzarem dos controls d'aquest tipus els quals els superposarem i un reproduirà el vídeo provinent de la càmera connectada a la Raspberry Pi i un altre l'àudio del micròfon connectat a la Raspberry Pi.

Inicialització

Per tal d'inicialitzar la llibreria haurem d'introduir el següent codi en la funció Main de l'arxiu Program.cs el qual és el punt d'entrada de la nostra aplicació. La funció Main quedaria així:

```
static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
```

```
        #Li donem a la llibreria el Path on estan les llibreries de VLC
        VlcContext.LibVlcDllsPath =
        CommonStrings.LIBVLC_DLLS_PATH_DEFAULT_VALUE_AMD64;

        #Li donem a la llibreria el Path on estan els plugins de VLC
        VlcContext.LibVlcPluginsPath =
        CommonStrings.PLUGINS_PATH_DEFAULT_VALUE_AMD64;

        #Configurem les següents opcions per tal de que no ens aparegui
        una finestra del símbol del sistema amb missatges de debig de
        l'aplicació
        VlcContext.StartupOptions.IgnoreConfig = true;
        VlcContext.StartupOptions.LogOptions.LogInFile = false;
        VlcContext.StartupOptions.LogOptions.ShowLoggerConsole = false;
        VlcContext.StartupOptions.LogOptions.Verbosity =
        VlcLogVerbosities.Debug;

        #Inicialitzar el context
        VlcContext.Initialize();

        #Posar en marxa el formulari de l'aplicació
        Application.Run(new Form1());

        #Si el formulari es tanca, tencar també el context de Vlc
        VlcContext.CloseAll();
    }
```

2.2.1.2. Mòduls

2.2.1.2.1. cSockets

Aquest mòdul conté les funcions i variables necessàries per crear un socket i atendre connexions a aquest. Fa us de la llibreria System.Net.Sockets de .NET.

Les variables i funcions implementades en aquest mòdul són les següents:

Variables

- **static ManualResetEvent allDone:**
  Variable que s'encarrega de resetejar el thread encarregat d'atendre peticions al socket i esperar fins que una connexió amb un client s'estableixi correctament.

- **Form1 form :**
  Variable que conté una referència al formulari de la nostra aplicació per tal de poder interactuar amb la interfície gràfica.

- **Socket handler :**
  Variable que es guarda una referència del socket que s'ha creat al connectar-se un client per així d'aquesta manera poder enviar una resposta més tard.

També hem creat una classe que representa l'estat d'una connexió amb un client i on s'emmagatzemen les variables necessàries per guardar l'estat.

```
public class StateObject
 {
        public Socket workSocket = null;
        public const int BufferSize = 1024;
        public byte[] buffer = new byte[BufferSize];
        public Form1 form = null;
        public StringBuilder sb = new StringBuilder();
        public string piso = null;
        public string puerta = null;
 }
```

Funcions

- **public void StartListening(Form1 form) :**
  Funció que es crida quan s'executa el programa i escolta peticions a la direcció IP de la interfície wifi i al port 1500 i configura adientment el socket de la connexió.

```
public void StartListening(Form1 form)
 {
        this.form = form;
        Socket listener = null;
        // Estableix el endpoint local per al socket.
        // Es demana la informació sobre la IP al DNS del PC
        IPHostEntry ipHostInfo = Dns.Resolve(Dns.GetHostName());
        IPAddress ipAddress = ipHostInfo.AddressList[0];
        IPEndPoint localEndPoint = new IPEndPoint(ipAddress,
          1500);

        // Es crea un socket TCP.
        listener = new Socket(AddressFamily.InterNetwork,
         SocketType.Stream, ProtocolType.Tcp);
```

```
            //S'enllaça el socket amb l'endpoint local i es comença a
              escoltar connexions entrants.
            try
            {
                listener.Bind(localEndPoint);
                listener.Listen(100);
                // Reseteja el thread.
                allDone.Reset();
                // Es posa en marxa un socket asíncron per atendre
                connexions entrants
                listener.BeginAccept(new AsyncCallback(AcceptCallback),
                    listener);

                //Esperar fins que la connexió es realitzi correctament
                allDone.WaitOne();

            }
            catch (Exception e)
            {
            }
    }
```

- **public void reListen ( ) :**
  Aquesta funció torna a posar en marza un socket asíncron per atendre connexions entrants

```
    public void reListen()
    {
        Socket listener  = null;
        allDone.Reset();
        listener.BeginAccept(new AsyncCallback(AcceptCallback),
        listener);

       allDone.WaitOne();
    }
```

- **public void AcceptCallback(IAsyncResult ar) :**
  Funció asíncrona que es crida quan un client es connecta.

```
public void AcceptCallback(IAsyncResult ar)
{
        allDone.Set();

        // Agafa el socket i atén la petició del client
        Socket listener = (Socket)ar.AsyncState;
        Socket handler = listener.EndAccept(ar);



        // Crea l'objecte d'estat
        StateObject state = new StateObject();
        state.workSocket = handler;
        //Crida  ala funció asíncrona per tal de llegir dades del client
        handler.BeginReceive(state.buffer, 0, StateObject.BufferSize,
            0,  new AsyncCallback(ReadCallback), state);
}
```

- **public void ReadCallback(IAsyncResult ar):**
  Funció asíncrona que s'encarrega de llegir dades del client.

```
public void ReadCallback(IAsyncResult ar)
{
        String content = String. Empty;

        // Agafa l'objecte d'estat i agafa el socket de connexió
        // de l'objecte d'estat
        StateObject state = (StateObject)ar.AsyncState;
        Socket handler = state.workSocket;

        // Llegeix data del socket del client
        int bytesRead = handler.EndReceive(ar);
```

```
if (bytesRead > 0)
{
    //Com possiblement arribaran més dades després
    concatenem l'string rebut amb l'String Builder de l'objecte
    estat
     state.sb.Append(Encoding.ASCII.GetString(state.buffer, 0,
    bytesRead));

    //Es mira si l'string rebut conté el caràcter separador
    content = state.sb.ToString();
    int pos = content.IndexOf("$");
    //Si es troba el caràcter separador i el pis encara no s'ha
     rebut
    if (pos > -1 && state.piso == null)
    {
          //Guardar el nom del pis
           state.piso = content.Substring(0, pos);
           state.sb.Remove(0, state.sb.Length);
           //Continuar rebent dades
          handler.BeginReceive(state.buffer, 0,
          StateObject.BufferSize, 0,  new
          AsyncCallback(ReadCallback), state);
    }
     //Si es troba el caràcter separador i la porta encara no s'ha
     rebut
    else if (pos > -1 && state.puerta == null)
    {
          //Guardar el nom de la porta
          state.puerta = content.Substring(0, pos);
          state.sb.Remove(0, state.sb.Length);
          //Guardar-nos el socket per després poder enviar
          this.handler = handler;
```

```
                //Cridar a una funció del formulari per tal de escriure
                les dades de pis i porta a un label. Com accedirem a un
                element de la interfície ho hem de fer des del thread
                principal.
                 form.Invoke((Action)delegate
                 {
                     form.llamar(state.piso, state.puerta);
                 });

            }
            //Si s'ha rebut l'string "RUN" (que vol dir que ja s'han
             iniciat les instàncies de ffmpeg en la Raspberry)
            else if (state.piso == null && state.puerta == null &&
                content == "RUN\0")
            {
                 state.piso = null;
                 state.puerta = null;
                 state.sb.Remove(0, state.sb.Length);
                 this.handler = handler;
                 //Reproduïr vídeo i àudio mitjançant el VlcControl del
                formulari
                 form.Invoke(new Action(() =>
                 {
                     form.Play();
                 }));
            }
            else
            {
                 // No totes les dades s'han rebut. Agafar més.
                 handler.BeginReceive(state.buffer, 0,
                 StateObject.BufferSize, 0,  new
                 AsyncCallback(ReadCallback), state);
            }
        }
    }
```

- **public void Send(String data) :**
  Funció que es crida des de el formulari i que envia la resposta al client que s'ha connectat prèviament.

```
public void Send(String data)
{
        // Convertir l'string a un vector de bytes
        byte[] byteData = Encoding.ASCII.GetBytes(data);

        // Començar a enviar al client
        handler.BeginSend(byteData, 0, byteData.Length, 0,
           new AsyncCallback(SendCallback), handler);
}
```

- **private static void SendCallback(IAsyncResult ar) :**
  Funció asíncrona que s'encarrega d'enviar les dades de resposta al client.

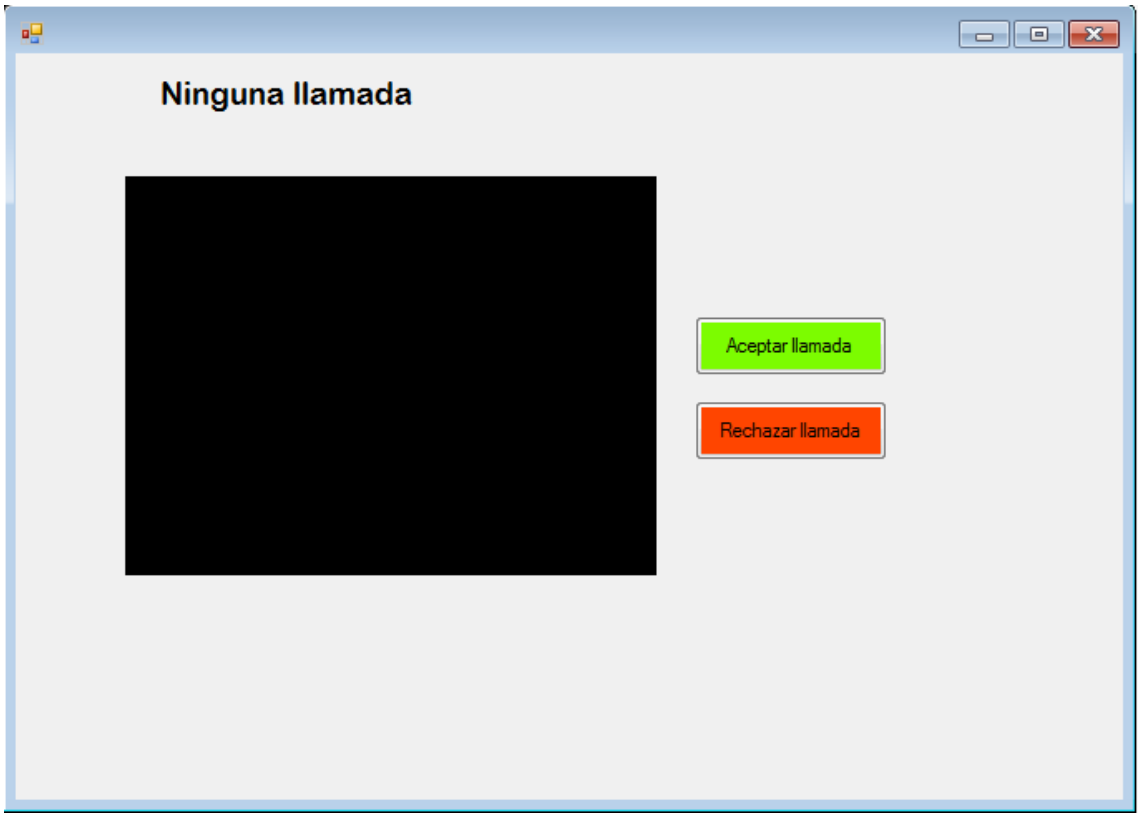
```
private static void SendCallback(IAsyncResult ar)
{
    try
    {
        //Agafar el socket de la connexió
        Socket handler = (Socket)ar.AsyncState;

        // Enviar les dades al client
        int bytesSent = handler.EndSend(ar);
        handler.Shutdown(SocketShutdown.Both);
        handler.Close();

    }
    catch (Exception e)
    {
    }
}
```

2.2.1.3. Formulari principal

La següent imatge mostra l'aparença del formulari de l'aplicació:



Les funcions que es criden quan es fa click a cadascun dels botons són les següents:

Botó "Aceptar llamada"

```
private void acceptButton_Click(object sender, EventArgs e)
 {
        //Parar el reproductor de so de la trucada
        player.Stop();
       //Des habilitar botó
        acceptButton.Enabled = false;
        //Enviar el caràcter 1 al client i tornar a escoltar peticions
        cSockets.Send("1");
        cSockets.reListen();
 }
```

Botó “Cancelar llamada”

```
private void cancelButton_Click(object sender, EventArgs e)
{
        //Tornar a posar el títol de “Ninguna llamada” al label i des
        habilitar botons
        title.Text = "Ninguna llamada";
        acceptButton.Enabled = false;
        cancelButton.Enabled = false;

        //Parar reproducció d’àudio i vídeo
        audio.Stop();
        video.Stop();

        //Aturar el procés que envia l’àudio del micro del PC, envia un
        caràcter 0 al client i torna a escoltar peticions
        ffmpeg.Kill();
        cSockets.Send("0");
        cSockets.reListen();
}
```

El codi de les crides “llamar” i “Play” que es criden des del mòdul cSockets es defineixen a continuació:

```
public void llamar(string piso, string puerta)
{
        //Posar el títol corresponent al label amb el pis i la porta
        title.Text = "Llamando a " + piso + " " + puerta + " ...";
        //Habilitar botons
        acceptButton.Enabled = true;
        cancelButton.Enabled = true;
        //Posar en marxa el programa que captura l’àudio del micròfon del
        PC
        startFFMPEG();
```

```
        //Reproduir un so de trucada
        playSound();
        myThread.Start();
}
```

```
public void Play()
{
        //Configurar les URL RTSP i començar a reproduir
        audio.Media = new LocationMedia
                                ("rtsp://192.168.1.39:5554/raspimicro");
        video.Media = new LocationMedia
                                ("rtsp://192.168.1.39:5554/raspicam");
        audio.Play();
        video.Play();
        title.Text = "Conexión establecida";
}
```

3. Pressupost

Els costos finals de manera resumida són els següents:

Costos hardware	
Element	**Preu**
Raspberry Pi	35 €
Raspberry Cam	30 €
Edimax Nano USB wifi	12,50 €
Micròfon de peu Logitech	15 €
Targeta SD de 16GB Sandisk	31,20 €
Polsador	1 €
Capsa per Raspberry Pi	7,50 €
Portàtil	600 €
Total	**732, 20 €**

Costos software

Per els costos software bàsicament tenim en compte les hores de desenvolupament , testeig i documentació destinades al projecte (veure secció de planificació temporal):

Número d'hores de desenvolupament: 422h
Preu per hora: 20 €
Total: 422h · 20 €/h = 8440€

4. Planificació temporal

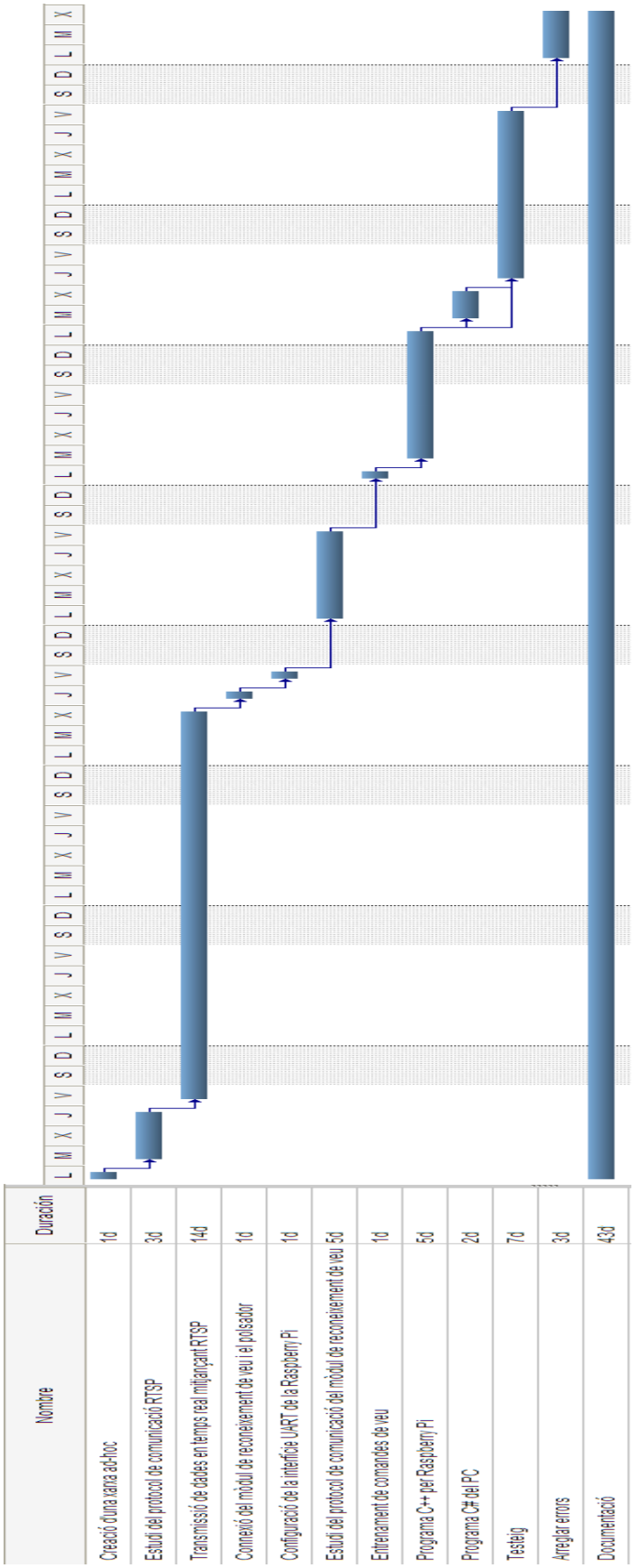
La descomposició del projecte en tasques per tal de duu a terme la planificació temporal ha sigut la següent:

1. Creació d'una xarxa ad-hoc. 8h
2. Estudi del protocol RTSP. 24h
3. Transmissió de dades en temps real mitjançant RTSP. 112 h
4. Connexió del mòdul de reconeixement de veu i el polsador. 4h
5. Configuració de la interfície UART de la Raspberry Pi. 4h
6. Estudi del protocol de comunicació del mòdul de reconeixement de veu. 40h.
7. Entrenament de comandes de veu. 8h
8. Programa C++ per Raspberry Pi. 40h
9. Programa C# per a PC. 16h
10. Testeig. 56h.
11. Arreglar errors. 24h
12. Documentació. 86h.

Total: 422 h

4.1. Diagrama de Gantt

El diagrama de Gannt corresponent a la planificació temporal és el següent:



Nombre	Duración
Creació d'una xarxa ad-hoc	1d
Estudi del protocol de comunicació RTSP	3d
Transmissió de dades en temps real mitjançant RTSP	14d
Connexió del mòdul de reconeixement de veu i el polsador	1d
Configuració de la interfície UART de la Raspberry Pi	1d
Estudi del protocol de comunicació del mòdul de reconeixement de veu	5d
Entrenament de comandes de veu	1d
Programa C++ per Raspberry Pi	5d
Programa C# del PC	2d
Testeig	7d
Arreglar errors	3d
Documentació	43d

5. Conclusions

Els objectius que teníem plantejats en un principi s'han assolit correctament. S'ha aconseguit transmetre perfectament imatges i àudio provinents de la placa microprocessador Raspberry Pi a un ordinador mitjançant una xarxa wi-fi ad-hoc creada per la mateixa placa.

De la mateixa manera s'ha aconseguit interconnectar un mòdul de reconeixement de veu i gravar comandes que posteriorment han estat perfectament reconegudes i integrades dins del sistema de reconeixement per control d'accessos.

5.1. Treball futur

Per tal de millorar el projecte es plantegen els següents objectius per tal de millorar-lo en un futur:

- **Dotar a la Raspberry Pi de connexió a Internet.** Amb això aconseguiríem:
  - Poder accedir a un webservice com Google Speech API que ens reconegués més eficientment la nostra veu.
  - Podríem comprovar en tot moment qui intenta accedir al nostre habitatge, ja que treballant conjuntament amb un dispositiu mòbil que també estigués connectat a internet podríem aconseguir aquest propòsit.
- **Incorporar un sensor de proximitat.** Per tal de que l'usuari no hagi de prémer un botó per tal de que comenci el reconeixement de veu, es podria incorporar un sensor de proximitat per tal de detectar quan una persona ha arribat.

5. Bibliografia

[1] RTP : audio and video for the Internet / Colin Perkins. Ed. Addison-Wesley. Disponible al catàleg UPC.

[2] Análisis de los protocolos en tiempo real en Ethernet RTP, RTCP y RTSP. Manuel Flores Vivas. Universidad de Málaga
http://www.slideshare.net/manuelfloresv/analisis-de-los-protocolos-de-tiempo-real-rtp-rtcp-y-rtsp

[3] Article a wikipedia sobre Real Time Streaming Protocol
http://es.wikipedia.org/wiki/Real_Time_Streaming_Protocol

[4] Pàgina oficial de la Raspberry Pi
http://www.raspberrypi.org

[5]Exemple d'us de la UART de la Raspberry Pi
http://www.raspberry-projects.com/pi/programming-in-c/uart-serial-port/using-the-uart

[6] Pàgina oficial del programa ffserver
http://www.ffmpeg.org/ffserver-all.html

[7] Pàgina oficial del programa ffmpeg
http://www.ffmpeg.org/ffmpeg-all.html

[8] Com crear una connexió ad-hoc
http://lcdev.dk/2012/11/18/raspberry-pi-tutorial-connect-to-wifi-or-create-an-encrypted-dhcp-enabled-ad-hoc-network-as-fallback/

[9] Com utilitzar els pins GPIO de la Raspberry Pi
https://sites.google.com/site/semilleroadt/raspberry-pi-tutorials/gpio

[10] Taula de les diferents funcions dels GPIO
http://elinux.org/RPi_BCM2835_GPIOs

[11] Manual del mòdul de reconeixement de veu EasyVR
http://download.tigal.com/veear/EasyVR_2/EasyVR_User_Manual_3.6.5.pdf

[12] Pàgina de Microsoft sobre .NET Windows Forms
http://msdn.microsoft.com/es-es/library/dd30h2yb(v=vs.110).aspx

[13] Pàgina de la llibreria wiringPi
http://wiringpi.com/

[14] Pàgina de la llibreria bcm2835
http://www.airspayce.com/mikem/bcm2835/